



# Алгоритмы и структуры данных

## Лекция 3. Методы разработки и анализа алгоритмов

---

Антон Штанюк (к.т.н, доцент)

5 февраля 2021 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева  
Институт радиоэлектроники информационных технологий  
Кафедра "Компьютерные технологии в проектировании и производстве"

Метод индукции

Прототипирование

Метод грубой силы

Жадные алгоритмы

Динамическое программирование

Перебор с возвратом

Измерение времени работы

Список литературы

В этой лекции мы рассмотрим некоторые вопросы, связанные с анализом и синтезом алгоритмов.

1. Метод индукции.
2. Прототипирование.
3. Метод "грубой силы".
4. Метод "разделяй и властвуй".
5. Жадные алгоритмы.
6. Динамическое программирование.
7. Перебор с возвратом.
8. Измерение времени работы.

## Метод индукции

---

Существует т.н. "принцип индукции", который гласит:

## **Принцип индукции**

**Наблюдение явления  $X$ , которое соответствует теории  $T$ , увеличивает вероятность того, что  $T$  - истинна.**

В математике применяется **метод математической индукции**, который чаще всего используют в утверждениях с числами. Суть этого метода в следующем:

Пусть требуется доказать утверждение  $S(n)$ , зависящее от целого числа  $n$ .

Доказательство разбивается на два этапа:

1. **Базис.** Показываем, что  $S(i)$  верно для  $i$ .
2. **Индуктивный переход.** Предполагаем, что  $n \geq i$  и доказываем, что из истинности  $S(n)$  следует истинность  $S(n + 1)$ .

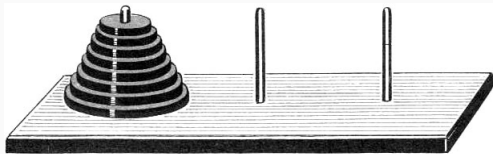
# Принцип домино

Доказательство по индукции наглядно может быть представлено в виде так называемого принципа домино. Пусть какое угодно число косточек домино выставлено в ряд таким образом, что каждая косточка, падая, обязательно опрокидывает следующую за ней косточку (в этом заключается индукционный переход). Тогда, если мы толкнём первую косточку (это база индукции), то все косточки в ряду упадут.



Е

сть три стержня и  $n$  колец разного размера. Класть можно только кольцо меньшего размера на кольцо большего размера. Можно ли переместить пирамидку с одного стержня на другой? Можно или нельзя?





- Пирамидку, в которой только одно кольцо  $n = 1$ , переместить можно (очевидно).
- Предположим, что мы умеем перемещать пирамидки с числом колец  $n \leq K$ .
- Попробуем научиться перемещать пирамидку с  $n = K + 1$ .  
Пирамидку из  $K$  колец, лежащих на самом большом  $K + 1$ -м кольце, мы можем согласно предположению переместить на любой стержень. Сделаем это, переместим её на третий стержень.
- Неподвижное  $K + 1$ -е кольцо не будет нам мешать провести алгоритм перемещения, так как оно самое большое.
- После перемещения  $K$  колец переместим оставшееся  $K + 1$ -е кольцо на второй стержень. Мы можем это сделать, так как второй стержень пустой.

- Теперь обратим внимание, тот факт, что второй стержень не пустой, не мешает нам класть на него любые кольца, так как имеющееся на нём кольцо самое большое (любое кольцо можно положить на большее, а значит и самое большое по условию задачи).
- И затем опять применим известный нам по предположению алгоритм перемещения  $K$  колец и переместим их на второй стержень, стержень с лежащим внизу  $K + 1$ -м кольцом. Таким образом, если мы умеем перемещать пирамидки с  $K$  кольцами, то умеем перемещать пирамидки и с  $K + 1$  кольцом.
- Следовательно утверждение верно для всех случаев, то есть для всех  $n$ .

## Прототипирование

---

Суть метода в том, чтобы написать упрощенную реализацию на простом языке, исследовать ее, а потом реализовать на нужном языке.

Например, можно реализовать вычисление факториала на языке **Scheme**, не используя систему типов:

```
(define (fact n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))))
```

## Метод грубой силы

---

Данный метод предполагает решение "в лоб", опирается на описание задачи и те данные, которые в нем приводятся.

Популярные алгоритмы:

- Умножение матриц по определению.
- Поиск наибольшего/наименьшего элемента в списке.
- Сортировка выбором (Selectionsort). □
- Пузырьковая сортировка (Bubblesort).
- Поиск подстроки в строке методом грубой силы.
- Поиск пары ближайших точек на плоскости.

Рассмотрим решение задачи поиска подстроки в строке.

```
int strstr(char* s, char* p)
{
    char *pbeg=p;
    while(*s) {
        if(*s==*p) {
            while(*s==*p) {
                s++;
                p++;
            }
            if(!*p || !(s-1))
                return 1;
            p=pbeg;
        }
        s++;
    }
    return 0;
}
```

## Метод "разделяй и властвуй"

---



1. Задача разбивается на несколько меньших экземпляров той же задачи.
2. Решаются сформированные меньшие экземпляры задачи (обычно рекурсивно).
3. При необходимости решение исходной задачи формируется как комбинация решений меньших экземпляров задачи.

Список алгоритмов:

- Сортировка слиянием (Mergesort) □
- Быстрая сортировка (QuickSort) □
- Бинарный поиск (Binary search) □
- Обход двоичного дерева (Treetraverse) □
- Решение задачи о поиске пары ближайших точек.

В качестве примера рассмотрим задачу вычисления квадратного корня. Мы можем определить функцию «квадратный корень» так:

$$\sqrt{x} = \text{такое } y, \text{ что } y \geq 0 \text{ и } y^2 = x$$

Как вычисляются квадратные корни? Наиболее часто применяется Ньютонов метод последовательных приближений, который основан на том, что имея некоторое неточное значение  $u$  для квадратного корня из числа  $x$ , мы можем с помощью простой манипуляции получить более точное значение (более близкое к настоящему квадратному корню), если возьмем среднее между  $u$  и  $x/u^2$ . Например, мы можем вычислить квадратный корень из 2 следующим образом: предположим, что начальное приближение равно 1.

Приближение	Частное $x/y$	Среднее
1	$\frac{2}{1} = 2$	$\frac{2 + 1}{2} = 1.5$
1.5	$\frac{2}{1.5} = 1.3333$	$\frac{1.3333 + 1.5}{2} = 1.4167$
1.4167	$\frac{2}{1.4167} = 1.4118$	$\frac{1.4167 + 1.4118}{2} = 1.4142$
1.4142	...	...

Продолжая этот процесс, мы получаем все более точные приближения к квадратному корню.

# Вычисление квадратного корня

```
#include <math.h>
#include <stdio.h>

double delta=0.000000001;

double average(double x,double y) {
    return (x+y)/2.0;
}
bool good(double guess,double x) {
    return abs(guess*guess-x)<delta;
}
double improve(double guess,double x) {
    return average(guess,x/guess);
}
```

```
double iter(double guess, double x) {
    if(good(guess,x))
        return guess;
    else
        return iter(improve(guess,x),x);
}
double calc(double arg) {
    return iter(1.0,arg);
}
int main() {
    double result=calc(2.0);
    printf("Sqrt(%lf)=%lf\n",2.0,result);
    return 0;
}
```

## Жадные алгоритмы

---



Это алгоритмы, принимающие на каждом шаге локально-оптимальное решение, предполагая, что конечное решение окажется оптимальным. В определенных случаях это приводит к правильному результату, а в других может приводить к неправильному.

## Задача о монетах

Монетная система некоторого государства состоит из нескольких монет разного достоинства. Требуется выдать сумму **S** наименьшим возможным количеством монет.

Для одних монетных систем решение находится правильное, для других - нет. Так, например, сумму в 24 копейки монетами в 1, 5 и 7 коп. жадный алгоритм разменивает так: 7 коп. — 3 шт., 1 коп. — 3 шт., в то время как правильное решение — 7 коп. — 2 шт., 5 коп. — 2 шт.

## Задача о черепашке

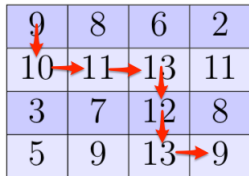
Дана двумерная таблица с числами. Черепашка, двигаясь из вернего левого угла, должна найти такой путь к правому нижнему углу, чтобы сумма чисел на пути была максимальна. Двигаться можно только вправо и вниз.

Рассмотрим следующую таблицу:

9	8	6	2
10	11	13	11
3	7	12	8
5	9	13	9

## Задача о черепашке

При поиске решения жадным алгоритмом мы предполагаем, что выбирая на каждом шаге максимальное число (справа или снизу), мы в итоге получим наилучший путь.



9	8	6	2
10	11	13	11
3	7	12	8
5	9	13	9

Приведенное решение оказалось верным.

А теперь рассмотрим такой случай:

9	8	6	2
10	11	13	11
3	7	12	8
<b>55</b>	9	13	9

При построении пути "локальным способом", он будет построен так же, как и в предыдущем примере, что приводит к неверному результату (мы обошли число 55).

Для правильного решения этой задачи нужно рассмотреть метод *динамического программирования*.

# Динамическое программирование

---

Это метод решения задач путем разбиения их на более простые подзадачи. Решение идет от простых подзадач к сложным, периодически используя ответы для уже решенных подзадач.

Типичный пример алгоритма: нахождение  $n$ -ого члена ряда Фибоначчи с использованием массива уже вычисленных значений.



Правильное решение задачи про черепаху.

Построим вторую матрицу, в которой будут содержаться локальные решения для любых клеток.

9	17	23	25
19	30	43	54
22	37	55	63
77	86	99	108

Теперь, в правом нижнем углу мы видим максимально возможное значение, которое может "собрать" черепашка. Двигаемся назад, к верхнему левому углу, выбирая максимальные значения.

9	8	6	2
10	11	13	11
3	7	12	8
5	9	13	9

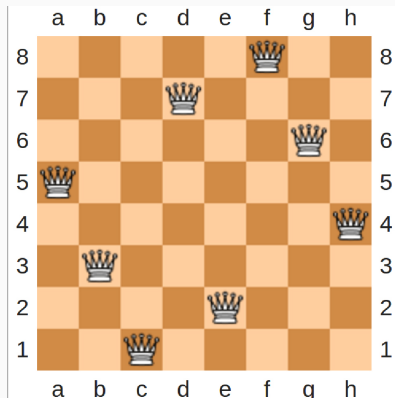
## Перебор с возвратом

---

Решение задачи о расстановки 8 ферзей на шахматной доске.

## Расстановка ферзей

Расставить на стандартной шахматной доске 8 ферзей таким образом, чтобы они не могли бить друг друга. Найти все возможные расстановки.



В более "математическом" виде задача может быть сформулирована несколькими способами, например, так: *Заполнить матрицу размером  $8 \times 8$  нулями и единицами таким образом, чтобы сумма всех элементов матрицы была равна 8, при этом сумма элементов ни в одном столбце, строке или диагональном ряде матрицы не превышала единицы.*

Конечная цель, поставленная перед решающим задачу, может формулироваться в нескольких вариантах:

- Построить одно, любое решение задачи.
- Аналитически доказать, что решение существует.
- Определить количество решений.
- Построить все возможные решения.

Одна из типовых задач по программированию алгоритмов перебора: создать компьютерную программу, находящую все возможные решения задачи.

Эта задача впервые была опубликована в 1850 году. Уже подсчитано, что на стандартном поле существует 4.426.165.368 различных комбинаций расположения фигур, но только 92 ведут к правильному решению.

Один из типовых алгоритмов решения задачи - использование поиска с возвратом: первый ферзь ставится на первую горизонталь, затем каждый следующий пытаются поставить на следующую так, чтобы его не били ранее установленные ферзи. Если на очередном этапе постановки свободных полей не оказывается, происходит возврат на шаг назад - переставляется ранее установленный ферзь.

## Измерение времени работы

---

Мы можем, используя средства стандартной библиотеки языка C, измерить время работы того или иного участка программы.

Нам понадобится заголовочный файл **time.h** и искусственный тип данных **clock\_t**.

До начала измеряемого фрагмента кода нужно запомнить время:

```
clock_t begin=clock();
```



После фрагмента нужно снова получить время:

```
clock_t end=clock();
```

И теперь необходимо найти разницу во времени:

```
double t1=(double)(end-begin)/CLOCKS_PER_SEC;
```

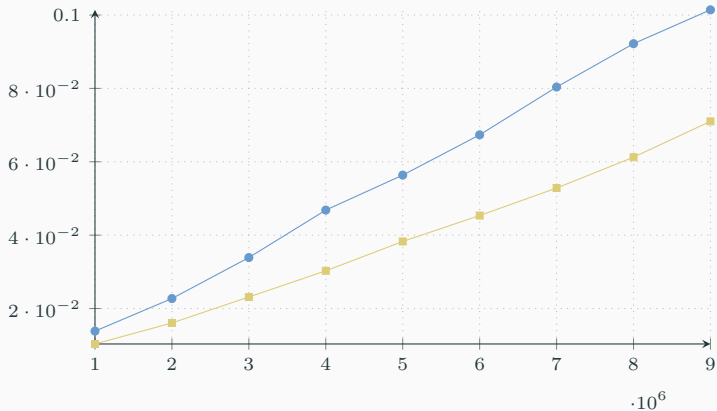
# Измерение времени работы

```
#include <stdio.h>
#include <time.h>
....
int main() {
    clock_t begin,end;
    double t1,t2;

    for(int n=1000000;n<=10000000;n+=1000000) {
        begin=clock();
        for(int i=0;i<n;i++)
            int res=factorial1(i%10);
        end=clock();
        t1=(double)(end-begin)/CLOCKS_PER_SEC;
        begin=clock();
        for(int i=0;i<n;i++)
            int res=factorial2(i%10);
        end=clock();
        t2=(double)(end-begin)/CLOCKS_PER_SEC;
        printf("%d,%lf,%lf\n",n,t1,t2);
    }
    return 0;
}
```






По результатом выполнения этой программы создадим файл с данными:






```
1000000,0.010340,0.013841  
2000000,0.016062,0.022723  
3000000,0.023146,0.033892  
4000000,0.030280,0.046831  
5000000,0.038278,0.056364  
6000000,0.045348,0.067345  
7000000,0.052841,0.080375  
8000000,0.061251,0.092191  
9000000,0.071022,0.101414  
10000000,0.076230,0.112484
```



## Список литературы

---

-  Кормен Т., Лейзерсон Ч., Ривест Р.  
*Алгоритмы: построение и анализ*  
МЦНМО, Москва, 2000
-  Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы:  
построение и анализ.  
*2-е изд. — М.: «Вильямс», 2006*
-  Википедия  
*Алгоритм*  
<http://ru.wikipedia.org/wiki/Алгоритм>
-  Википедия  
*Список алгоритмов*  
[http://ru.wikipedia.org/wiki/Список\\_алгоритмов](http://ru.wikipedia.org/wiki/Список_алгоритмов)
-  Традиция  
*Задача коммивояжёра*  
<http://traditio.ru/wiki/Задача>

-  Википедия  
*NP-полная задача*  
<http://ru.wikipedia.org/wiki/NP-полная>
-  Серджвик Р.  
*Фундаментальные алгоритмы на C++. Части 1-4*  
Diasoft, 2001
-  Седжвик Р.  
*Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск*  
СПб.: ДиаСофтЮП, 2003
-  Седжвик Р.  
*Фундаментальные алгоритмы на C. Алгоритмы на графах*  
СПб.: ДиаСофтЮП, 2003
-  Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.  
*Издательский дом «Вильямс», 2000*



Кнут Д.

*Искусство программирования, том 1. Основные алгоритмы*  
3-е изд. — М.: «Вильямс», 2006



Кнут Д.

*Искусство программирования, том 2. Получисленные методы*  
3-е изд. — М.: «Вильямс», 2007



Кнут Д.

*Искусство программирования, том 3. Сортировка и поиск*  
2-е изд. — М.: «Вильямс», 2007



Кнут Д.

*Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений*  
М.: «Вильямс», 2007





Кнут Д.

*Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации*

М.: «Вильямс», 2007